Third Pole Inc,

CLI Architecture document

ACUTE system

Author: Hazem Kabbara

Table of Contents

[Executive Summary 3](#_Toc155606205)

[System Description 3](#_Toc155606206)

[System Design 4](#_Toc155606207)

[CLI Paser 6](#_Toc155606208)

[Creating a Menu 7](#_Toc155606209)

[Navigation and Prompt Update 7](#_Toc155606210)

[Serial Port configuration 11](#_Toc155606211)

# Executive Summary

This document describes the command line interface design for use in the Acute submodules (SPM, GDNs, GSA and VIM). The intension is to allow the user to debug, configure, calibrate each module to ease the development of the Acute system.

# System Description

The embedded command line interpreter, cli, allows applications to bind a directory of commands and the associated functions to execute those commands. The features of cli are lightweight, flexible, modular, and cross-platform.

command line interpreter (cli) provides a directory hierarchy of commands similar in concept to the Linux directory structure. There exists the concept of a root directory and subdirectory.

Subdirectories can be created under root or under other subdirectories. Commands can be created under root or any subdirectory.

When the user enters a command, the matching command structure is located from the current directory and the associated call-back function is invoked to execute the command.

When the user enters a directory name to change directories, the cli will change to that directory and update the prompt accordingly to display its tree position. This new directory becomes the present working directory (pwd).

Application call backs are invoked with the (argc, \*argv[]) parameters. argc is the number of parameters. argv[0] is the command string while argv[1,2,...,ARGC\_MAX] are command specific parameters.

When the user enters a command, the matching command structure is located from the registered commands and the associated function executed.

There are several built-in commands for directory navigation:

h help

~ moves to the top directory, root

.. moves the directory up one level

ls global list of commands

? command help

There are several built in directories that applications can register for organization:

cli\_config\_dir - Commands to manage configuration.

cli\_show\_dir - Commands to show internals: configuration, data, etc.:

cli\_clear\_dir - Commands to clear counters, runtime data, etc.

cli\_opertion\_dir - Commands to manage the runtime operation.

cli\_debug\_dir - Commands to facilitate debugging.

Public Functions:

cli\_mkcmd - add a command to a current or root directory

cli\_mkdir - add a subdirectory to a specified directory

cli\_engine - command processing engine

cli\_init - initialization

# System Design

A screenshot of a computer

Description automatically generated

Figure - CLI Design Flowchart

The CLI command structure is a link list of cli\_record\_t which contains the following data:

typedef struct t\_nodal\_record {

struct t\_nodal\_record \*cmd\_list;

char name[CLI\_ENTRY\_LEN+1];

char desc[CLI\_ENTRY\_DESC\_LEN+1];

menu\_prio priority;

cli\_funcvar\_t fv;

struct t\_nodal\_record \*parent\_dir;

struct t\_nodal\_record \*next\_entry;

struct t\_nodal\_record \*link2subdir;

} cli\_record\_t;

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Data Name** | **Description** |
| t\_nadal\_record | cmd\_list | It is a pointer to a command structure |
| char | name [] | Command name |
| char | desc [] | Command description |
| menu\_prio | priority | Command priority |
| cli\_funcvar\_t | fv | Command callback function |
| t\_nadal\_record | parent\_dir | Pointer to the current parent directory structure |
| t\_nadal\_record | next\_entry | Pointer to the next command entry structure |
| t\_nadal\_record | link2sudir | Pointer to the subdirectory when subcommand is required |

For every project, menu\_builder.c file will construct the desired menu by calling series of calls to cli\_mkcmd() and cli\_mkdir() functions which will have the following context respectively:

/\* Add a command \*/

extern RC\_CLI\_t cli\_mkcmd(

char \*name,

char \*desc,

menu\_prio prio,

cli\_funcvar\_t fv,

cli\_record\_t \*dir\_rec,

cli\_record\_t \*cmd\_rec);

/\* Add a directory \*/

extern RC\_CLI\_t cli\_mkdir (

char \*name,

char \*desc,

menu\_prio prio,

cli\_record\_t \*dir\_rec,

cli\_record\_t \*sub\_dir);

As noticed, the cli\_mkcmd() function has a callback function, whereas cli\_mkdir() function does not. It is a subcommand and not yet considered as complete to invoke the callback function. Thus, the last part of the command structure must be terminated by command (cli\_mkcmd).

The user will need to modify uart\_theard0\_entry.c file to redirect serial to the emulator terminal. See file for details.

# CLI Paser

The CLI parser is the function cli\_engine(), which receives the command, and navigates through the command link list trying to find a match. When a match occurs, the associated callback function is called passing the arguments (argv[0], argv[1],..). The callback function tests the validity of these arguments. If type mismatch like expect a number and received a string or the number is out of range, the callback function will have to display a message informing the user the expected value options or valid range.

The following callback function illustrates what was described:

static void cli\_led1 (uint32\_t argc, char \*argv[]){

int value;

bsp\_leds\_t leds;

R\_BSP\_LedsGet(&leds);

char thisStr[64];

if ((!strcmp(argv[1], "?")) || (!strcmp(argv[1], "\0"))) {

sprintf(thisStr, "\r\nOptions are 0 = OFF and 1 = ON \r\n "); //🡨 provides a feedback to the user

printThis(thisStr);

return;

}

else {

if(is\_number(argv[1])) {

value = atoi(argv[1]);

if(value == 0) {

g\_ioport.p\_api->pinWrite(leds.p\_leds[0], IOPORT\_LEVEL\_HIGH);

sprintf(thisStr, "\r\nLED1 is turned OFF \r\n");

printThis(thisStr);

return;

}

else if (value == 1) {

g\_ioport.p\_api->pinWrite(leds.p\_leds[0], IOPORT\_LEVEL\_LOW);

sprintf(thisStr, "\r\nLED1 is turned ON \r\n");

printThis(thisStr);

return;

}

else {

sprintf(thisStr, "\r\nInvalid argument \r\n"); //<- argument is not a number

printThis(thisStr);

return;

}

}

}

printf ("\r\nInvalid Argument!!\r\n");

}

# Creating a Menu

In the MenuBuilder.c file, Menu\_Build(), create your own menu and submenu as desired. The following example is used for illustration:

char menu\_build (void) {

#ifdef CLI\_ROOT\_DIRS

RC\_CLI\_t rc;

/\* Main Menu Commands \*/

rc = cli\_mkdir("gpio", "GPIO management", mon, NULL, &cli\_gpio\_dir);

rc = cli\_mkdir("can", "CAN management", mon, NULL, &cli\_can\_dir);

rc = cli\_mkdir("adc", "ADC channel management", mon, NULL, &cli\_adc\_dir);

rc = cli\_mkdir("dac", "DAC channel management", mon, NULL, &cli\_dac\_dir);

rc = cli\_mkdir("fan", "FAN channel management", mon, NULL, &cli\_fan\_dir);

rc = cli\_mkdir("power", "Command to manage subsystem power", mon, NULL, &cli\_power\_dir);

rc = cli\_mkcmd("sysname", "Display system name", mon, &cli\_sysname, NULL, &cli\_sysname\_cmd);

rc = cli\_mkcmd("alarms", "Display oustansding alarms", mon, &cli\_alarms, NULL, &cli\_alarms\_cmd);

/\* GPIO Sub Commands \*/

rc = cli\_mkcmd("led1", "Manage LED1 (ON, OFF)", mon, &cli\_led1, &cli\_gpio\_dir, &cli\_led1\_cmd);

rc = cli\_mkcmd("led2", "Manage LED2 (ON, OFF)", mon, &cli\_led2, &cli\_gpio\_dir, &cli\_led2\_cmd);

rc = cli\_mkcmd("led3", "Manage LED3 (ON, OFF)", mon, &cli\_led3, &cli\_gpio\_dir, &cli\_led3\_cmd);

return (char)rc;

#endif

}

In the above example, the callback functions are found in the same file.

# Navigation and Prompt Update

The default prompt is set in cli.c around line 142 (char device\_prompt[48] = "XXX";)

When the user connects to the CLI for the first time, the prompt is defaulted to the submodule name. For example, if the submodule is SPM, the prompt will show as follows:

XXX >>

When a question mark “?” is typed and followed by a carriage return, the system will show the available command/subcommands as follows:

A screenshot of a computer

Description automatically generated

When the user per say wants to manage the GPIO led pin, the user types gpio as follows:

A screenshot of a computer

Description automatically generated

As shown the prompt is updated to show the menu is one level down in gpio submenu. If the user type “?” and carriage return the options are displayed as shown below:

A screenshot of a computer program

Description automatically generated

If the user typed led1 ?, it will show what arguments led1 takes as shown below:

A screenshot of a computer program

Description automatically generated

Now to turn LED1 off type “led1 0” and to turn on type “led1 1” as shown below:

A computer screen shot of a black screen

Description automatically generated

To go back one level type “..” followed by a carriage return as follows:

A computer screen shot of a black screen

Description automatically generated

# Serial Port configuration

Using TeraTerm, set your configuration as follows:

A screenshot of a computer

Description automatically generated

In addition, terminal setup option, turn on echo as follows:

A screenshot of a computer

Description automatically generated